CoSine IP Service Delivery Platform Application Architecture

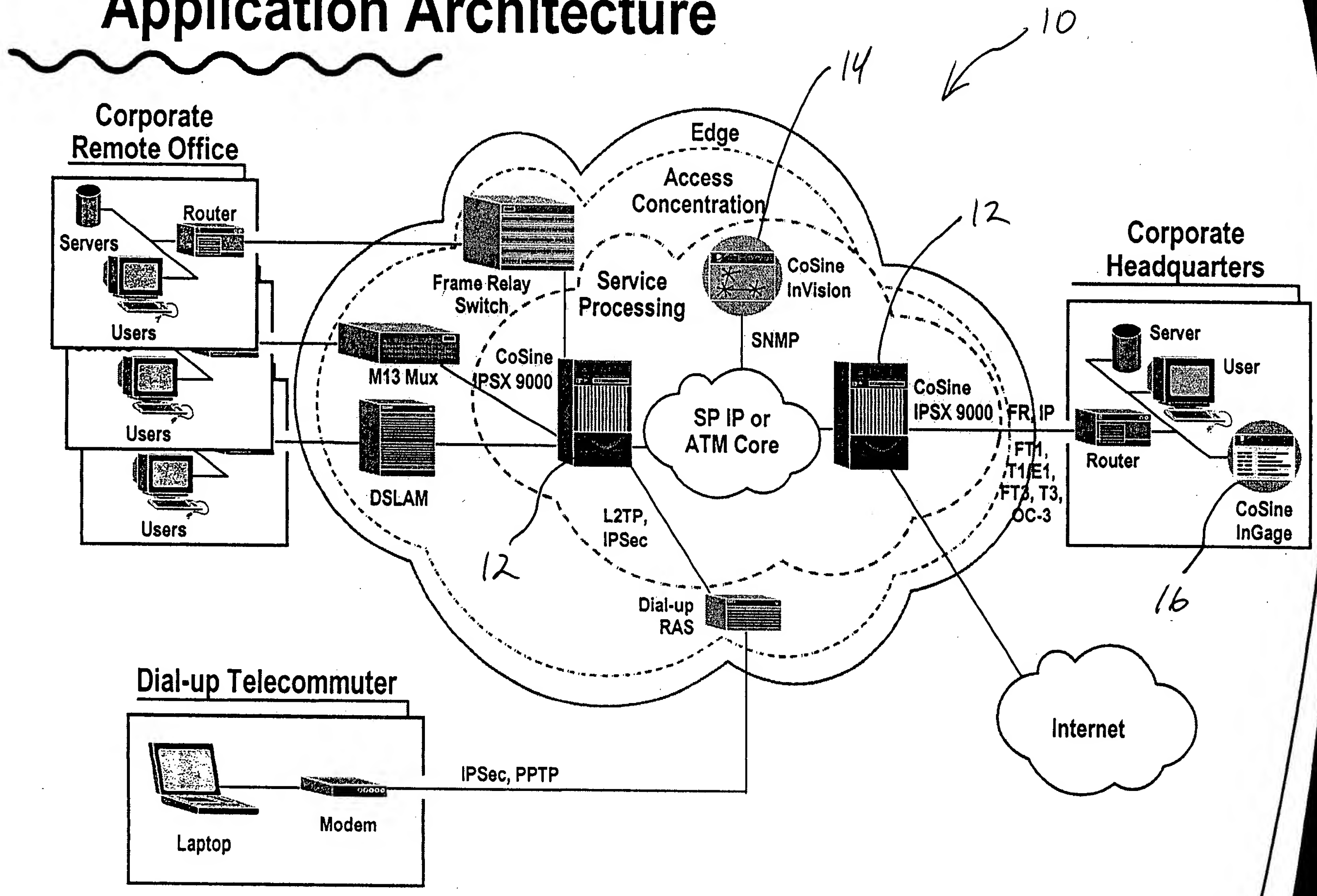
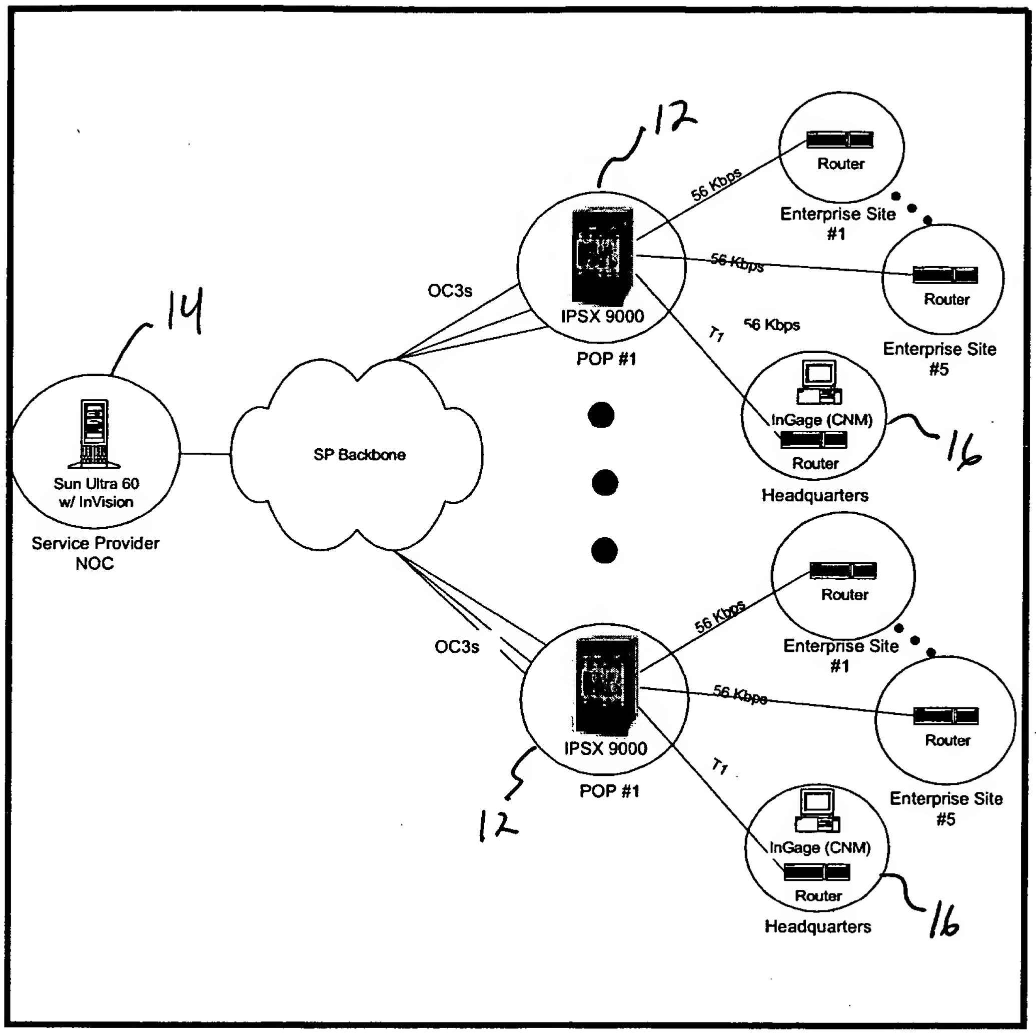


Figure 1

Diagram 5 — Managed Firewall Service with CoSine's Network-based Solution



POP Infrastructure

The POP access infrastructure in the network-based managed firewall service model is based on the CoSine Communications IPSX 9000 Service Processing Switch. The base configuration for the switch includes:

- 26-slot chassis
- Redundant power supply
- IPNOS Base Software
- Ring Bridge & Ring Bridge Pass-Thru (to complete midplane)
- Control Blade (for communications with InVision Services Management System)
- Dual-port Channelized DS3 Access Blade
- Dual-port Unchannelized DS3 Access Blades
- Processor Blade
- OC-3c POS Trunk Blade

The following tables analyze the cost structure of all of the above models and projects these costs out over 5 years:

Figure 2

IPNOS Overview

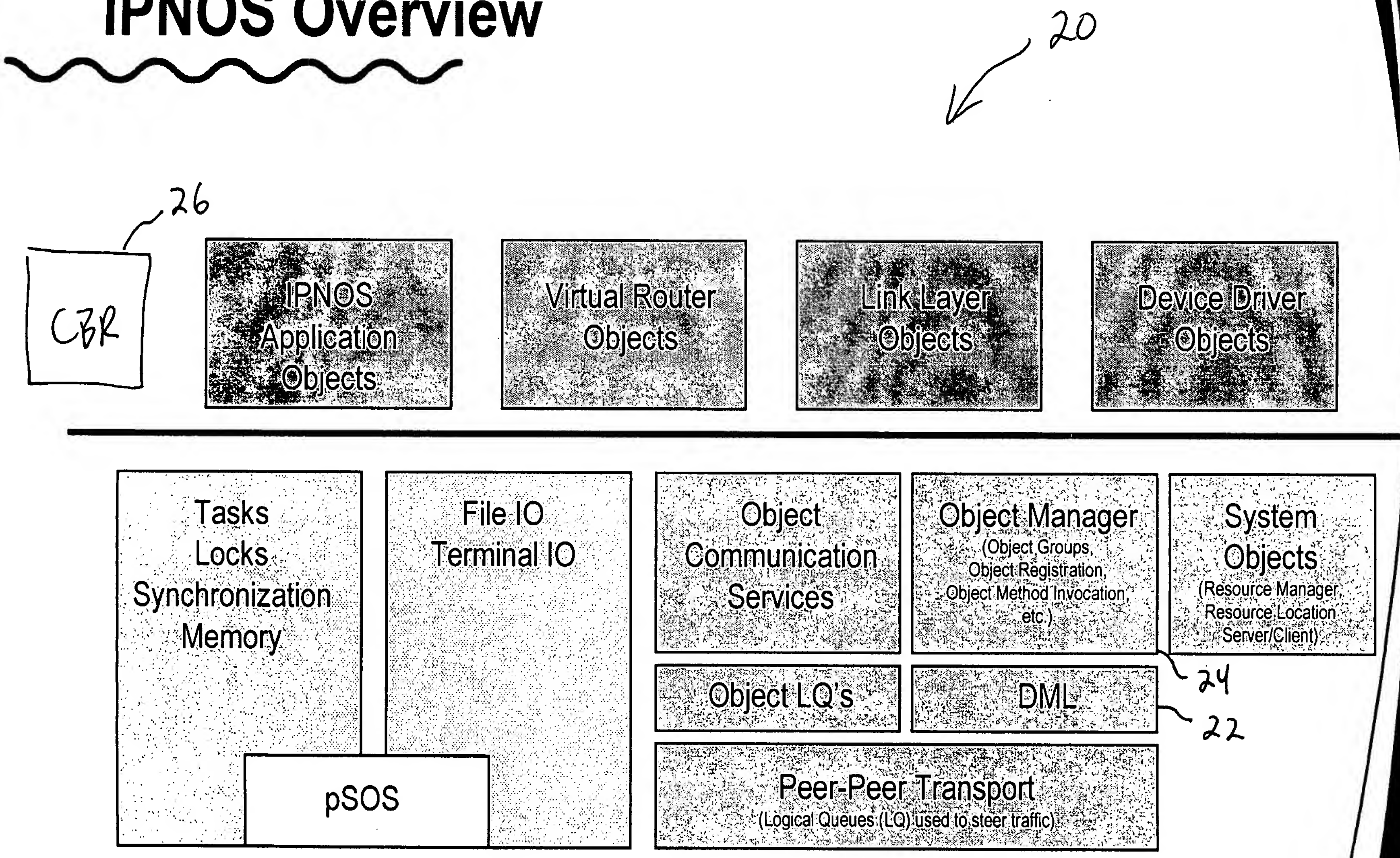

20
26
CBR
IPNOS Application Objects
Virtual Router Objects
Link Layer Objects
Device Driver Objects
Tasks
Locks
Synchronization
Memory
pSOS
File IO
Terminal IO
Object Communication Services
Object Manager
(Object Groups, Object Registration, Object Method Invocation, etc.)
System Objects
(Resource Manager, Resource Location Server/Client)
24
Object LQ's
DML
22
Peer-Peer Transport
(Logical Queues (LQ) used to steer traffic)

Fig. 3

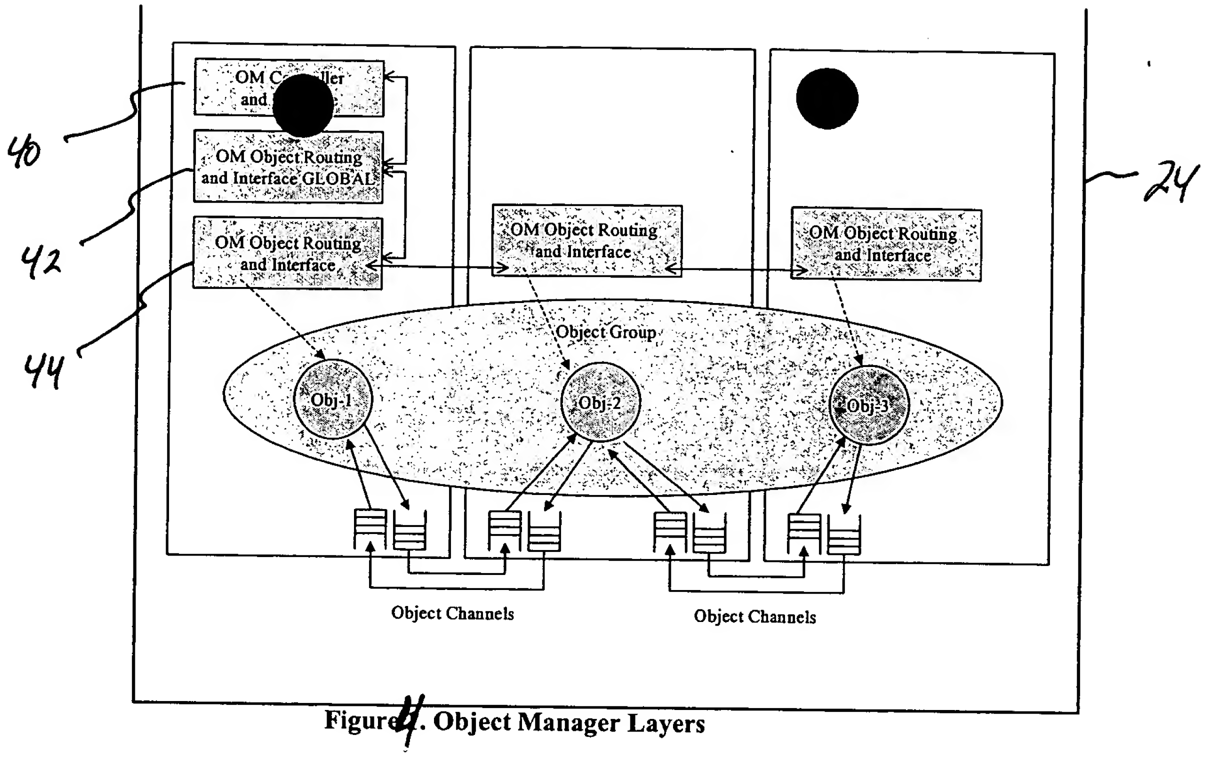

OM C ller
and
OM Object Routing and Interface GLOBAL
OM Object Routing and Interface
OM Object Routing and Interface
OM Object Routing and Interface
Object Group
Obj-1
Obj-2
Obj-3
Object Channels
Object Channels
40
42
44
24

Figure 4. Object Manager Layers

IPSX object database consists of two types of database: Global, managed on Master Control Blade by OMORIG and distributed local databases, managed by OMORI agents on every PE present in the system. Global database is a superset of the extracts from local databases.

2.2. Object

Objects represent a basic unit of management for purposes of fault tolerance, computational load balancing etc. One or more adjacent protocol modules can be placed into a single object. It is also possible that a module is split across two objects.

2.3. Group

Group is an aggregation point for all objects that comprises the VR. Group and VR have one-to-one mapping. A Group encompasses objects, which are located in different address spaces. Group Id, which identifies a group, is unique in the scope of a single IPSX system.

2.4. Object Class

Figure 5 shows the distinction between an Object Class and an Object Group. Both are collections of objects. As shown in Figure 2, an object class is a set of objects that have the same type signature and behavior (e.g. Applications Class, TCP/IP Class and Interfaces Class). In contrast, for an object group, the constituent objects do not necessarily have the same type signature and behavior (e.g. Object Groups 1 to 3). There can be multiple objects of the same class in an object group (e.g. Object Group 2 has two objects of Interface Class). On the other hand, an object group need not have an object of each class (e.g. Object Group 3 does not have an object of Interface Class).

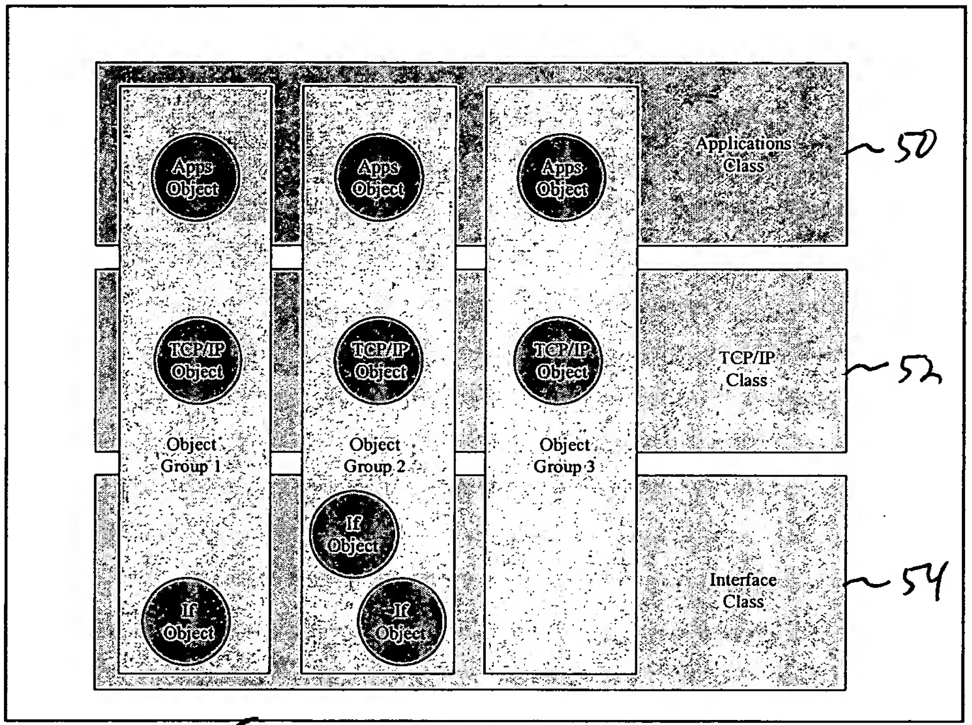


Figure 5 Object Classes and Groups.

2.5. OMCD and OMORIG

OMCD is the agent, which interfaces to the Configuration Manager. As shown on Figure 3 OMCD manages

- Global list of VPN in the IPSX system
- List of VRs per VPN

The caveats for VPN and VRs are:

- VPN ID equal to 0 is illegal;
- Global uniqueness of VPN ID across IPSX systems is the responsibility of the Service Management System (SMS).

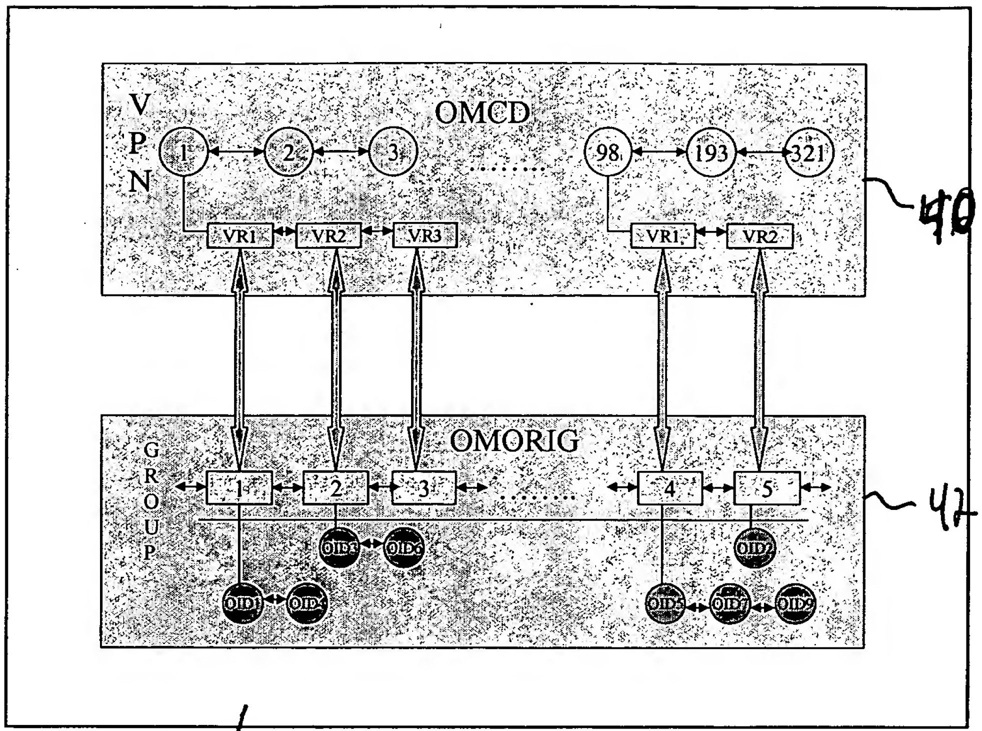


Figure ~~3~~ 6 OMCD and OMORIG Database maps.

OMCD creates vpn descriptor every time when Configuration managers request VPN creation. Every VPN is identified by unique VPN ID.

Virtual Router (VR) is identified by VR ID, which is the IP Address of the VR. VR ID is unique in the VPN context. When Configuration Manager requests creation of an existing VR in the VPN, VR creation request is rejected. Otherwise vr descriptor will be created.

There are several types of the VR:

- ISP (Internet Service Provider) VR: Typically there is 1 such VR for a single IPSX.
- Control VR: There can be only one Control VR for a single IPSX. This VR is used to host the management applications such as SNMP, Telnet etc.
- Customer VR: There are several Customer VRs in a single IPSX. Typically, there is 1 Customer VR per customer service point.

Detailed VR creation process is described below

OMORIG agent runs on every Control Blade, whether it is Master or Standby Blade. OMORI local sends the change only to Master. Control Blade Redundancy feature, described in [4] takes care of replicating and synchronizing OMORIG database from Master to Standby.

2.6. OMORIG Object ID Links

OMORIG provides several mappings of Object Ids. It manages list of object Ids, which

- Are located on the same address space.
- Belong to the same group
- Sorted Global object ID list
- Unsorted Global object ID list

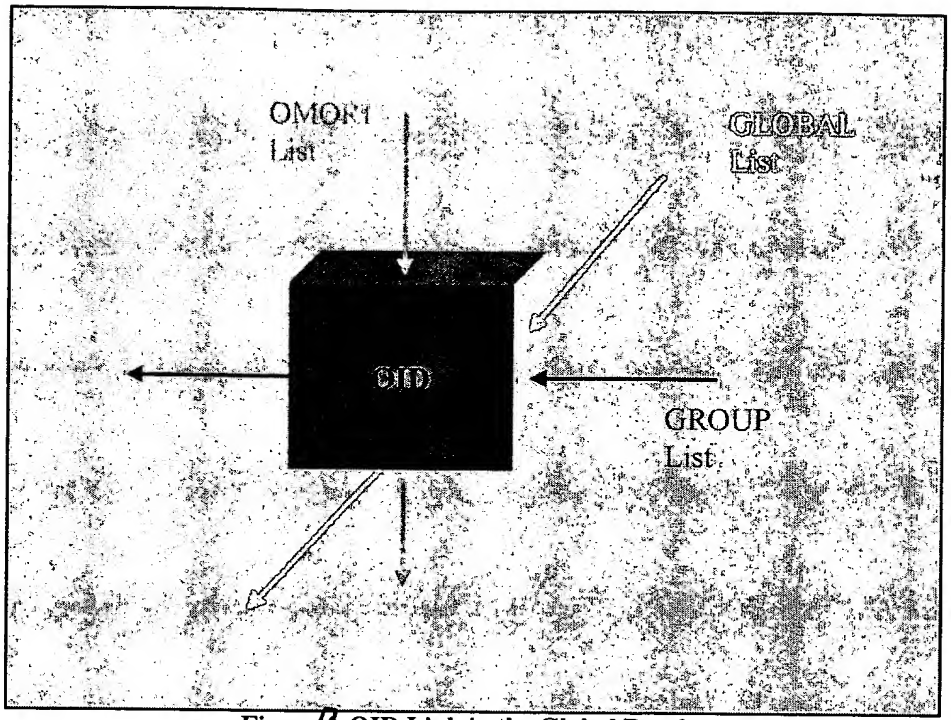


Figure 7. OID Link in the Global Database

OID link data structure and OMORIG API are published in page 40.

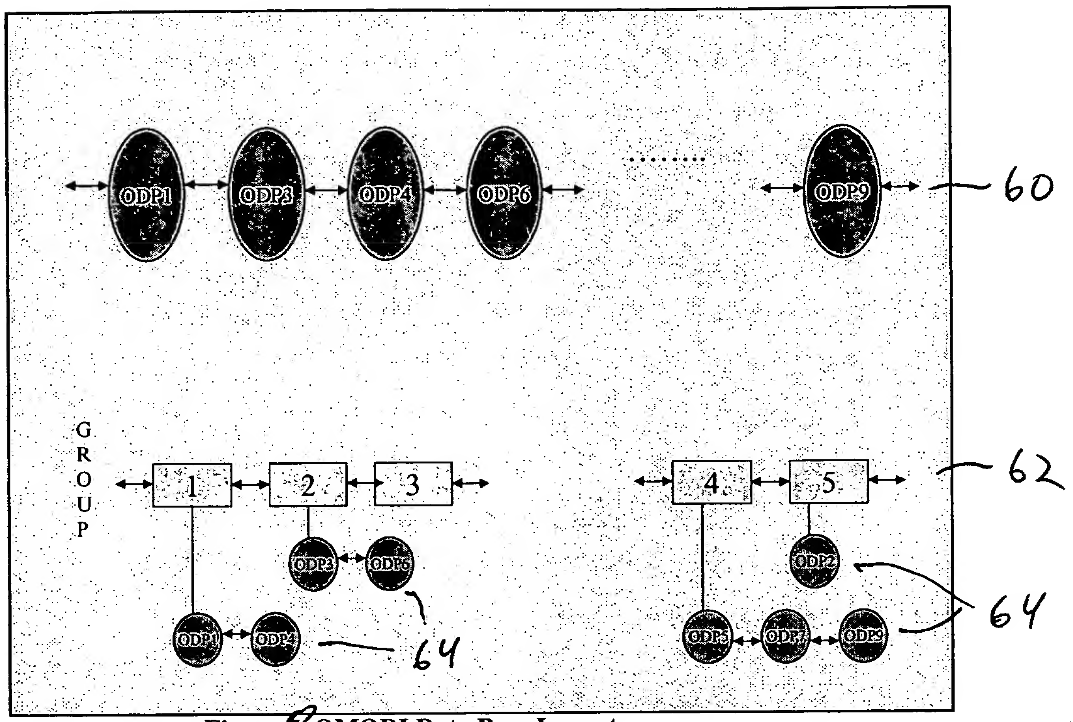


Figure 8 OMORI Data Base Layout

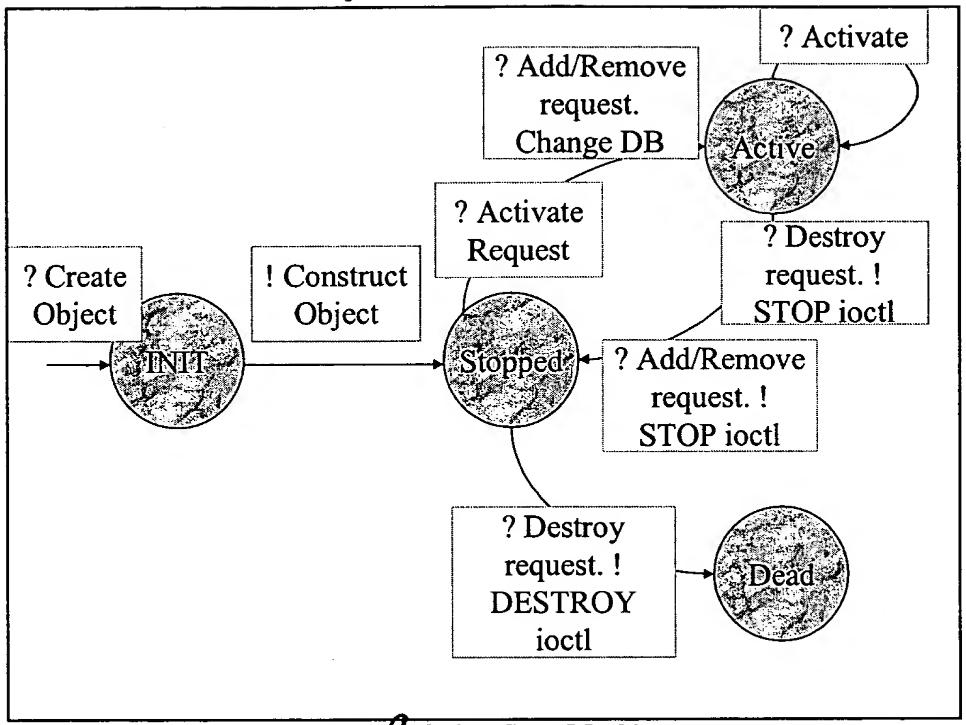

? Activate
? Add/Remove request. Change DB
Active
? Activate Request
? Destroy request. ! STOP ioctl
? Create Object
! Construct Object
INIT
Stopped
? Add/Remove request. ! STOP ioctl
? Destroy request. ! DESTROY ioctl
Dead

Figure 9. Object State Machine

3. OM as a DML Application

Distributed Messaging Layer (DML) is used to provide inter-processor communication and isolated channels for data and control messages. Detailed explanation on DML can be found in [3]. OMORIG and OMORI communicate via predefined DML channel DML_CHAN_DATA. All IPNOS nodes in the system are members of DML_CHAN_DATA. During initialization process OMORI register to DML receive function, which will be called every time a new packet arrives on DML_CHAN_DATA. OMORIG and OMORI are DML applications and therefore they are notified on every dynamic event in the system.

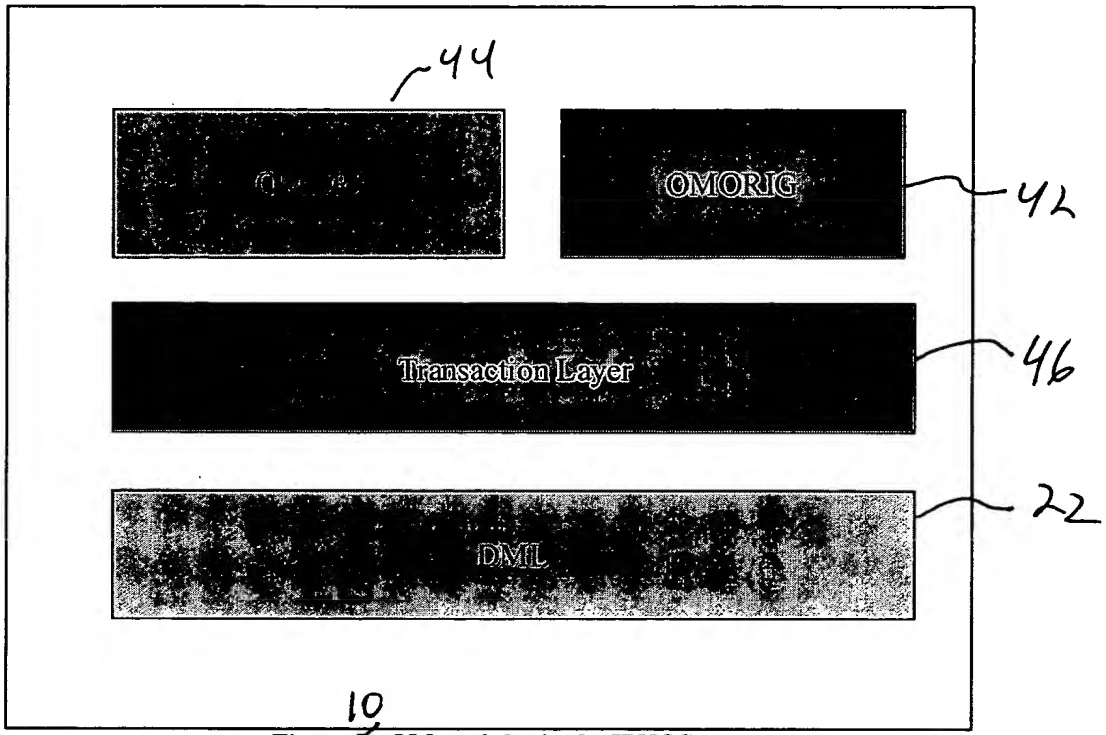


Figure 1. OM modules in the IPNOS.

3.1. Dynamic events

There are four types of dynamic events indicated by DML. These are:

- Peer Up – new IPNOS node detected and reachable.
- Peer Down – existing IPNOS node became unreachable
- Master Up – new Master elected in the system
- Master Down – existing Master became unreachable

On peer down event OMORI agent aborts all the pending transactions associated with the peer, which went down.

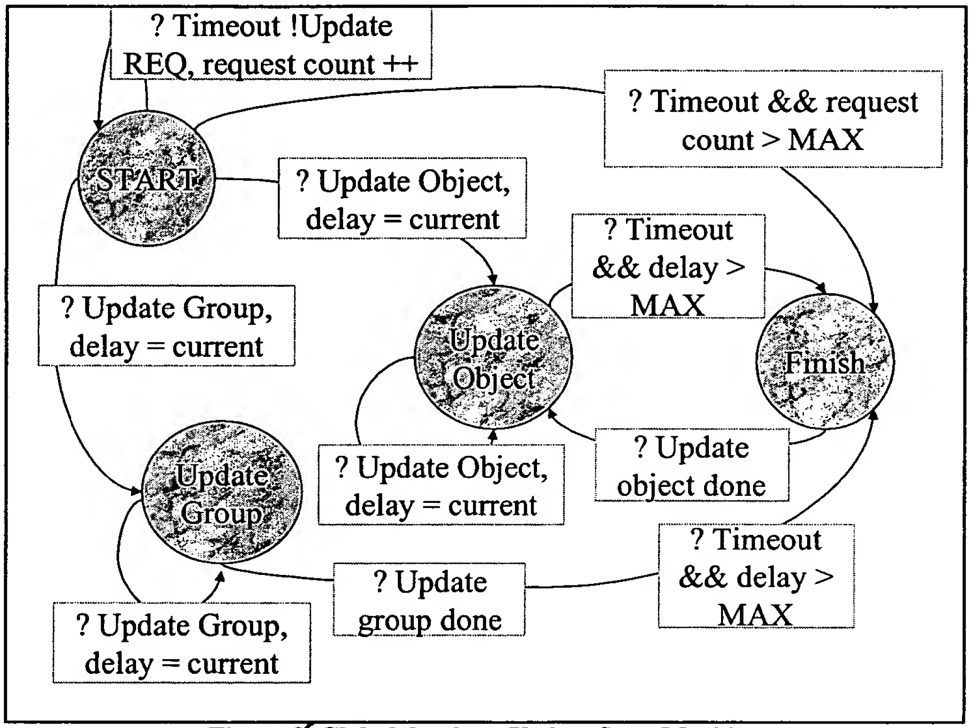
3.1.2. OMORIG

On peer down event OMORIG destroys in its database all the objects, which are local to peer which went down. After that a scrub of the database is done. This includes destroying all groups, which do not have any objects in them and destroying VR associated with the group..

On peer up event and master up event OMORIG agent runs global database update protocol described in the section 3.1.3.

3.1.3. Update protocol

On peer up event OMORIG agent initiates a database update for local objects of the new peer. OMORIG maintains state machine per OMORI. Global Database Update State Transition Diagram is shown in Figure 8 and a detailed description of the transitions is in Table 2.



? Timeout !Update REQ, request count ++
? Timeout && request count > MAX
START
? Update Object, delay = current
? Timeout && delay > MAX
? Update Group, delay = current
Update Object
Finish
? Update object done
? Update Object, delay = current
Update Group
? Timeout && delay > MAX
? Update group done
? Update Group, delay = current

Figure ~~8~~. Global database Update State Machine

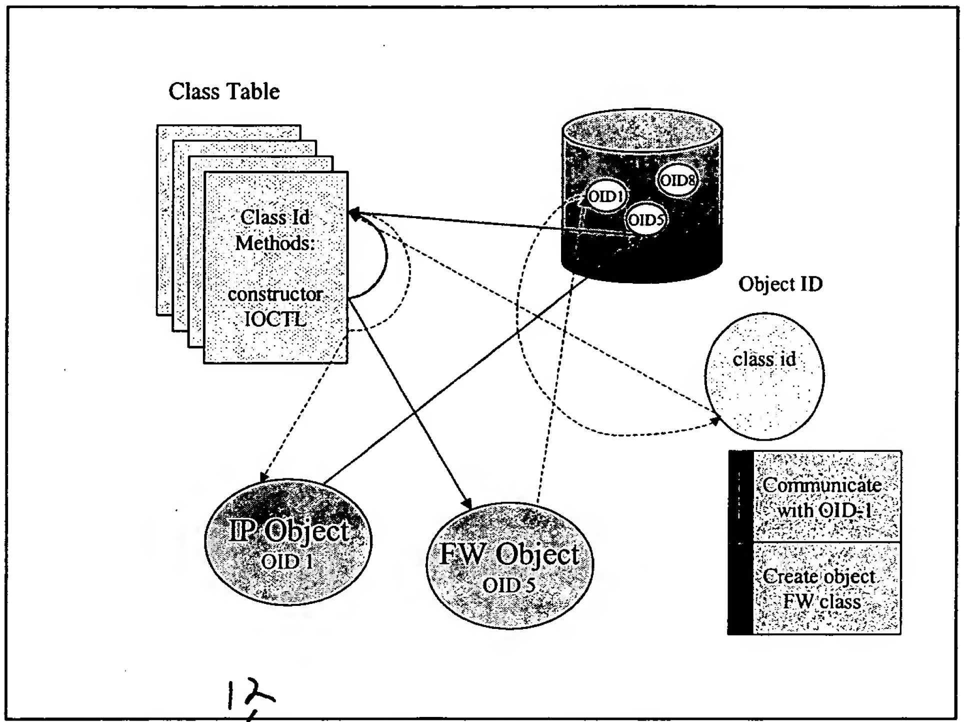
11

STATE	EVENT	ACTION
START	TIMEOUT && (request count < MAX)	Send update request
START	TIMEOUT && (request count > MAX)	Peer did not reply. Update FAILED Transit to FINISH state.
START	RECV UPDATE GROUP message	Transit to UPDATE GROUP state. Set last update equal to the current time.
START	RECV UPDATE OBJECT message	Transit to UPDATE OBJECT state. Set last update equal to the current time.

4. Objects: Creation and Inter Communication

4.1. Overview

In Figure 9 the communication for object creation is shown. IP object with OID 1 requests Firewall object to be created. OM creates object descriptor and based on the specified class of the object (e.g. Firewall), OM finds the appropriate *constructor* function in the object class table and constructs the new object. The same algorithm is used for management communications between objects (IOCTL). Based on the class id of the destination object appropriate *control* function from the object class table is called. It is the responsibility of the object implementers is to supply handlers for all supported IOCTLs.



12

Figure ~~9~~. Object Creation and Communication.

The IOCTL mechanism is described in section 4.3. Typically IOCTL between objects is used for Management Plane communications. For Data Plane communications between objects, object to object channels are used. Object channels are described in the section 4.5.

4.2. Creation

Objects can be created in three different ways:

- **REGISTERED**: Created as system comes up (e.g. drivers) and registered to the OMORI with object id, having physical location meaning.
- **CREATED BY OM**: Created by Object Manager. In this case OMORI creates a locally unique (in the scope of this address space) object ID that in conjunction with address space id gives unique object id inside of the system. To create an object, the constructor function based on the object class will be called.

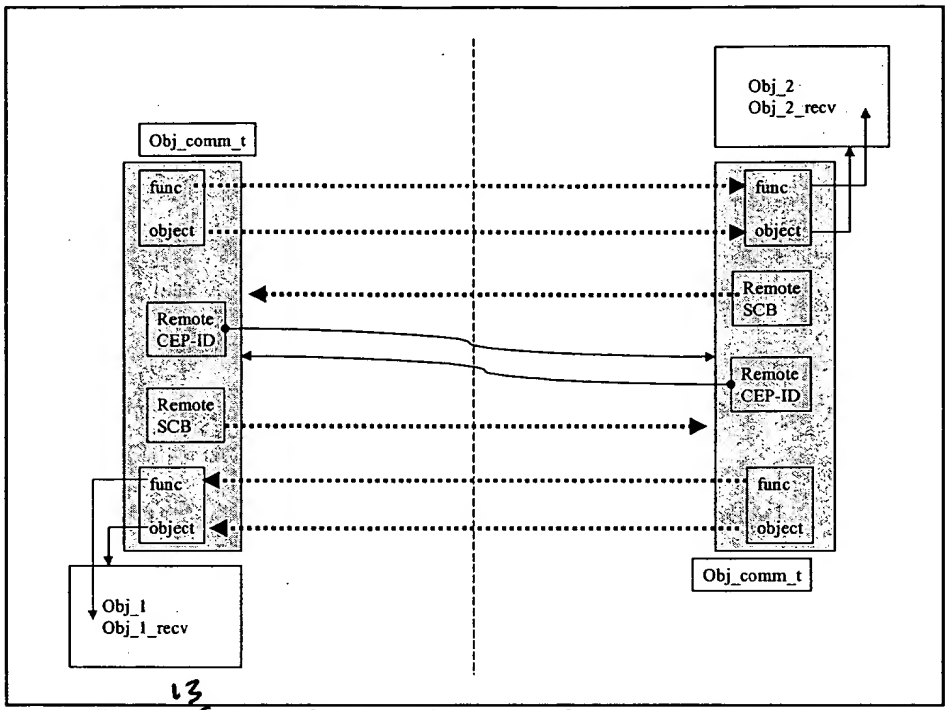


Figure ~~10~~ 13. Object Channels: Connecting CEPs in same address space

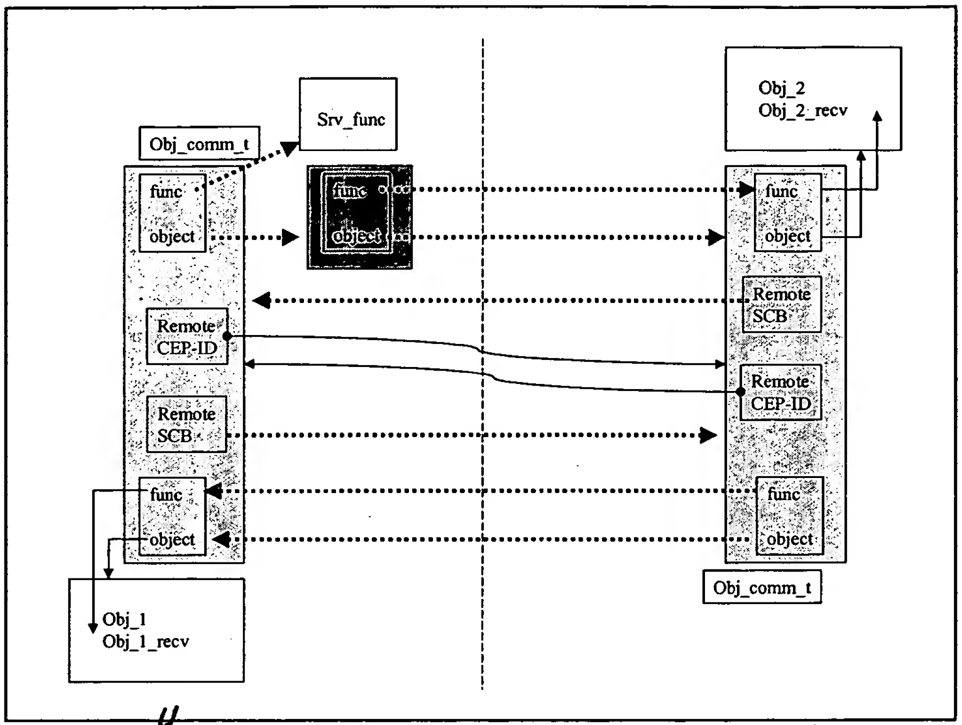


Figure 1~~1~~4: Object Channels: Pushing service on Transmit side of Object-1

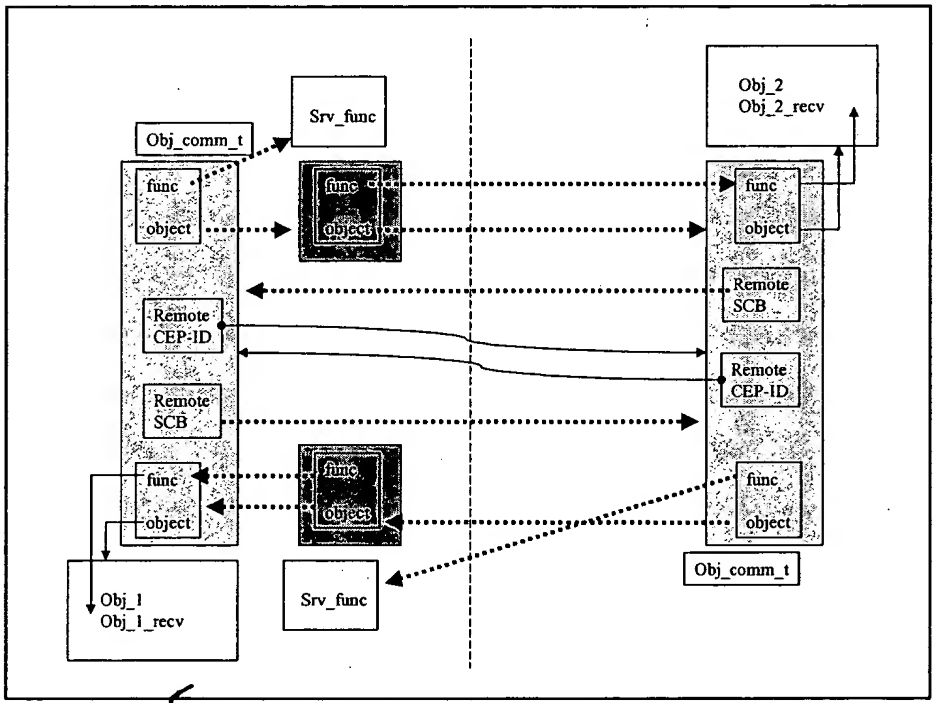


Figure 1~~2~~5: Object Channels: Pushing service on Receive side of Object-1

Step	Local CEP Object	Local IPNOS	Local RM/ LQ	Remote RM/ LQ	Remote IPNOS	Remote CEP Object
1	`obj_associate_channel(local_chan, local_cep_id, remote_cep_id)`					
2		*/* Allocate remote LQ */* resmng_alloc_resource (RESOURCE_DATA_CONNECTION, 0, remote_cep_id -> object.address_space_id, &remote_lq)				
3				**Lookup resource tag and allocate from *remote* LQ**		
4		*/* Ask remote LQ to allocate local LQ */* status = omori_obj_ioctl_by_id (&remote_lq, remote_lq.group, OBJ_CTL_CODE_ANY (LQUSER_BIND), &lq_bind, sizeof (lq_bind)); memcpy (&local_lq,				

		&lq_bind.lq_object.local, sizeof (object_id_t));				
5				**Use resmng_alloc_resource() to allocate *local* LQ**		
6			**Lookup resource tag and allocate from *local* LQ**			
7				**Return allocated *local* LQ**		
8		***/* Bind Local and Remote LQs */*** status = omori_obj_ioctl_by_id (&local_lq, local_lq.group , OBJ_CTL_CODE_ANY (LQUSER_BIND), &lq_bind, sizeof (lq_bind));				
9			**Setup LQ-API parameters to point to *remote* LQ**	**Setup LQ-API parameters to point to *local* LQ**		
10		***/* Push local LQ as a service onto local channel */*** status = omori_obj_ioctl_by_id				

09/893,956

Fig. 16b

		(&local_lq, local_lq.group , OBJ_CTL_CODE_ANY (LQUSER_BIND), &lq_bind, sizeof (lq_bind));				
11	**Lookup CEP address**					
12					***/* Push remote LQ as a service onto remote channel */*** status = omori_obj_ioctl_by_id (&remote_lq, remote_lq.group, OBJ_CTL_CODE_ANY (LQUSER_BIND), &lq_bind, sizeof (lq_bind));	
13						**Lookup CEP address**

Fig. 16c

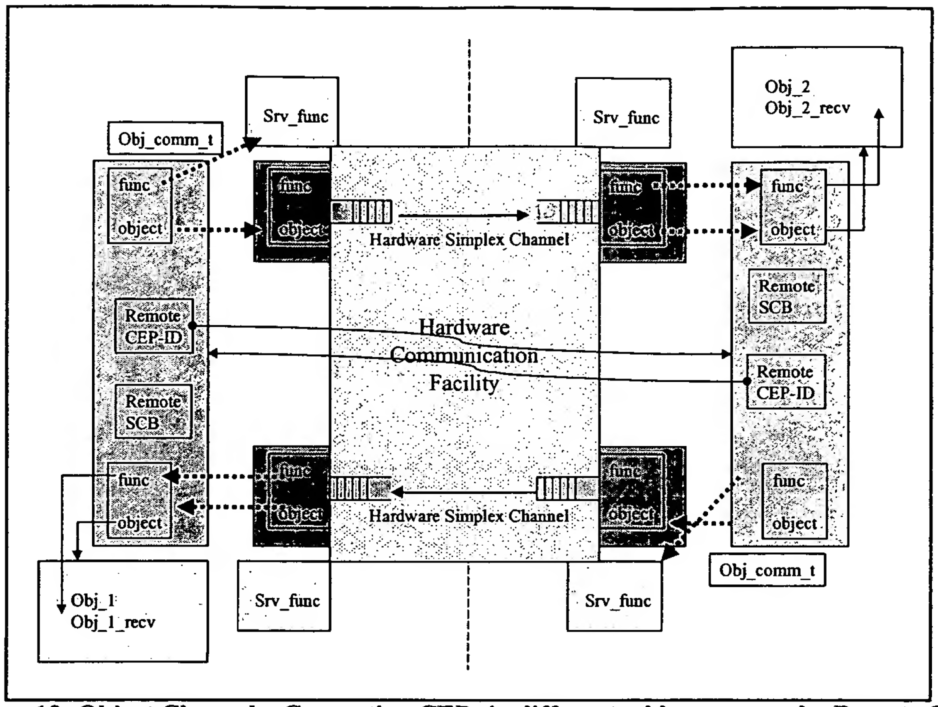


Figure ~~13~~ 17: Object Channels: Connecting CEPs in different address spaces by Remote Channel Service

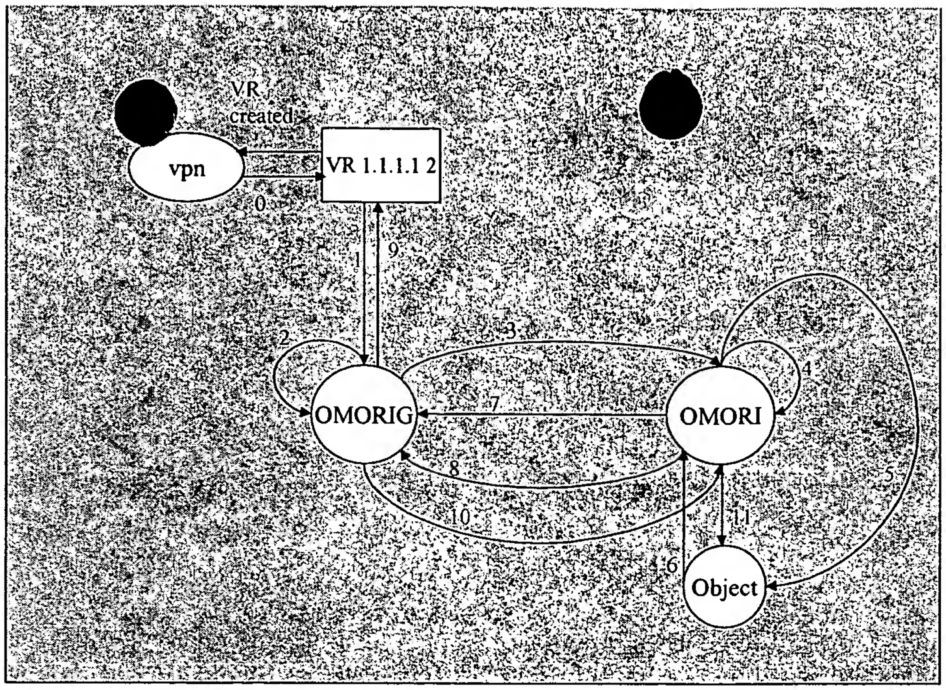


Figure ~~15~~ 16 VR Creation with Single object

09663483.091300

2		Create group 1; create object id link of selected class. Validate address space id on capability to service specified object class. Send request CREATE_OBJECT to capable OMORI (2). Wait for OMORI reply.		
3			Receive CREATE_OBJECT request for specified group. Lookup a group; not found. Create group 1; Create object descriptor of selected class. Call the class constructor.	
4		Receive MV_OBJ_TO_GROUP request; add object id to OMORIG Database	add object to the group, send MV_OBJ_TO_GROUP request to OMORIG	
5				Create and initialize an object; return SUCCESS or FAILURE.
6			If FAILURE remove object from the group, send MV_OBJ_TO_GROUP and RM_OBJ_FROM_GROUP to OMORIG; ================ Else send reply for CREATE_OBJECT request to OMORIG	
7		Receive MV_OBJ_TO_GROUP request, move object to the group 0(OM_BASE_GROUP); Receive RM_OBJ_FROM_GROUP request; remove object id from OMORIG		

Fig. 19b

		Database ============		
8		Receive Object CREATE reply. Signal to OMCD		
9	VR created, return status to user			
10		Send ACTIVATE object message to OMORI (2)		
11			Receive ACTIVATE object message. For all the objects of this group send generic IOCTL ACTIVATE_OBJECT	Activate object, Do object-specific action to make it active, operational

Fig. 19c

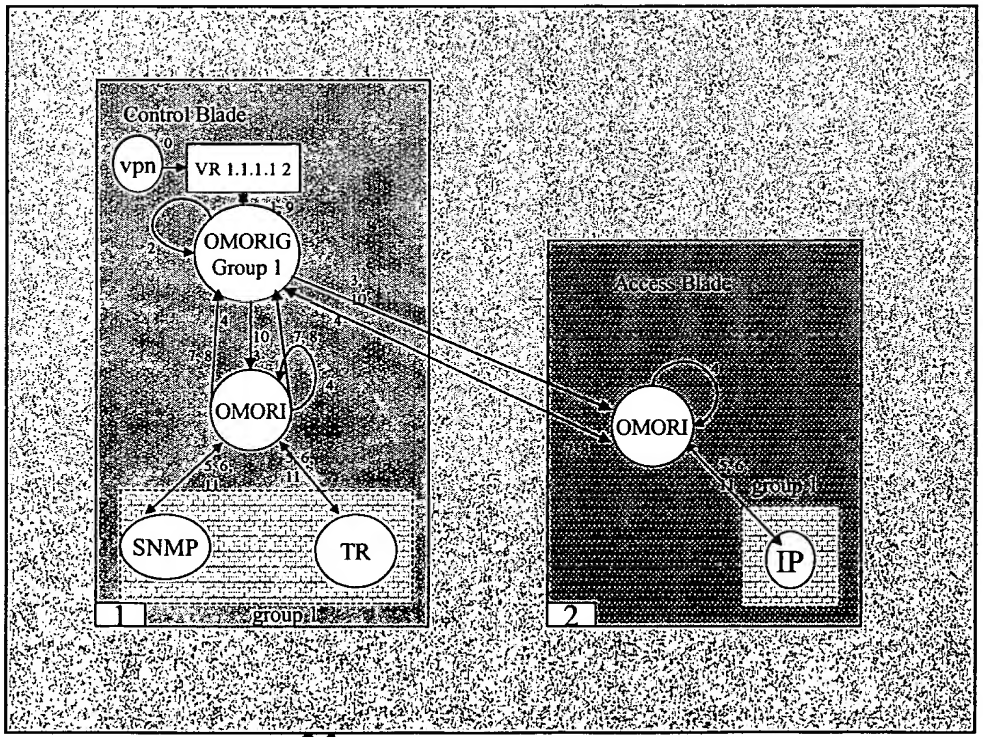


Figure 20 VR Creation with Multiple Objects.

Step	OMCD	OMORIG	OMORI	Object

Fig 21a-

0	Create unique vr_descriptor_t for specified VPN, fills with default value and adds VR to the list of VR per VPN.			
1	Requests group creation for specified VR with class_group_selector on specified address space.			
2		Create group 1; create object id link of selected class. Validate address space id on capability to service specified object class. Send request CREATE_OBJECT to capable OMORIs (1 and 2). Wait for reply from both OMORIs.		
3			Receive CREATE_OBJECT request for specified group. Lookup a group; not found. Create group 1; Create object descriptor of selected class. Call the class constructor.	
4		Receive MV_OBJ_TO_GROUP request; add object id to OMORIG Database	add object to the group, send MV_OBJ_TO_GROUP request to OMORIG	
5				Create and initialize an object; return SUCCESS or FAILURE.
6			If FAILURE remove object from the group, send MV_OBJ_TO_GROUP and RM_OBJ_FROM_GROUP to OMORIG;	

0965348Z.091501

Fig 21a

			============== Else send reply for CREATE_OBJECT request to OMORIG	
7		Receive MV_OBJ_TO_GROUP request, move object to the group 0(OM_BASE_GROUP); Receive RM_OBJ_FROM_GROUP request; remove object id from OMORIG Database ==============		
8		Receive Object CREATE reply. If all the object replied then signal to OMCD, otherwise do nothing		
8	VR created, return status to user			
10		Send ACTIVATE object message to every OMORI (1,2) where objects were created		
11			Receive ACTIVATE object message. For all the objects of this group send generic IOCTL ACTIVATE_OBJECT	
12				Activate object, Do object-specific action to make it active, operational

Fig 21b.

5.2. VR Deletion

5.2.1. Multiple Objects

Compliment operation to create VR with multiple objects in the group is to destroy such a VR. Destroy VR operation is shown on the Figure 16. Sequence of steps, describing this is in the Table 7.

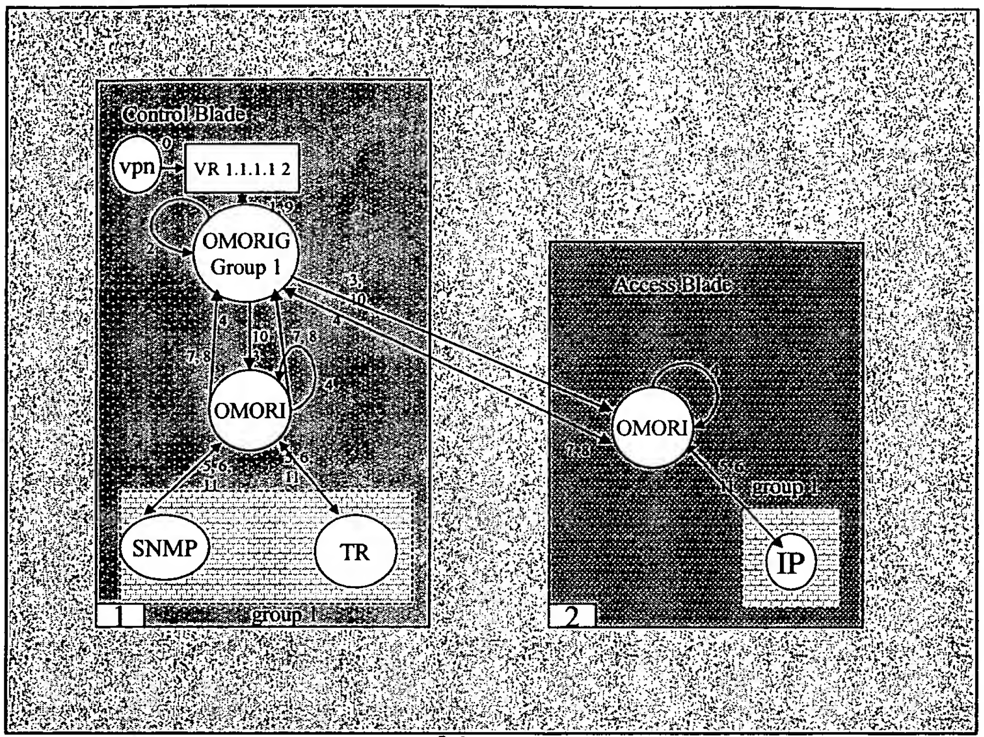


Figure 22. VR Deletion

Step	OMCD	OMORIG	OMORI	Object
0	For specified VPN and VR lookup vr_descriptor. Call OMORIG to delete corresponding group.			
1		Lookup group descriptor by specified id. Filter OMORIs which have objects to be destroyed(which belong to the specified group)		
2			Receive DESTROY_GROUP_ OBJECTS request for specified group. Lookup a group; Send generic IOCTL STOP_OBJECT to every local object, which belongs to the group	
3				Stop operating, destroy all connections with other objects

Fig. 23a

4			Send generic IOCTL DESTROY_OBJECT to every local object, which belongs to the group	
5				Free itself
6			If FAILURE remove object from the group, send MV_OBJ_TO_GROU P and RM_OBJ_FROM_GR OUP to OMORIG;	
7		Receive MV_OBJ_TO_GROUP request, move object to the group 0(OM_BASE_GROUP); Receive RM_OBJ_FROM_GROU P request; remove object id from OMORIG Database		
8		Receive Object DESTROY_GROUP_OBJ ECTS. Subtract number of destroyed objects from the total number of objects in the group (VR). If all objects destroyed then signal to OMCD, otherwise do nothing.		
8	VR destroyed, return status to user			

Fig. 23b

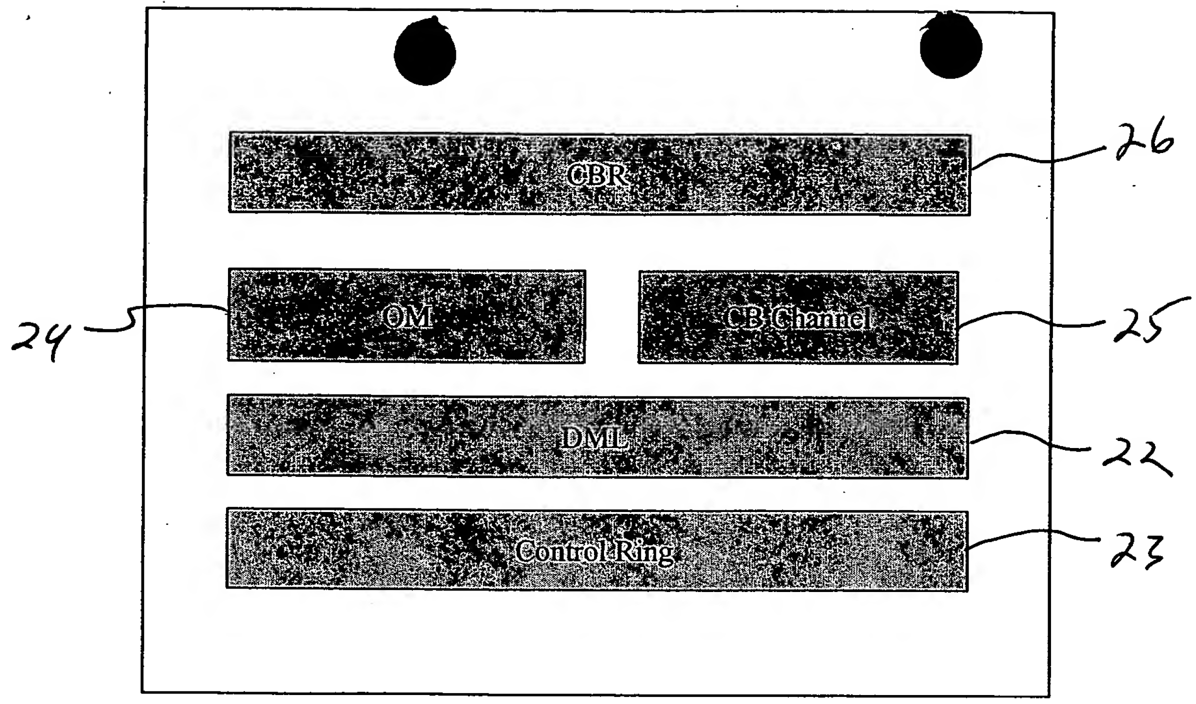


Fig. 24

~~2. CBR Design~~

In one embodiment Control Blade redundancy (CBR) 26 is designed to create and maintain replicas of the Master Control Blade management information on the Standby Control Blades and reuse ~~it in~~ that information in the case of failure of the current Master and election of a new Master. The Control Ring normally elects the new Master. If the Control Ring detection mechanism fails a software-based leader election protocol implemented by DML will elect the new Master. This redundancy is illustrated in Figure 25.

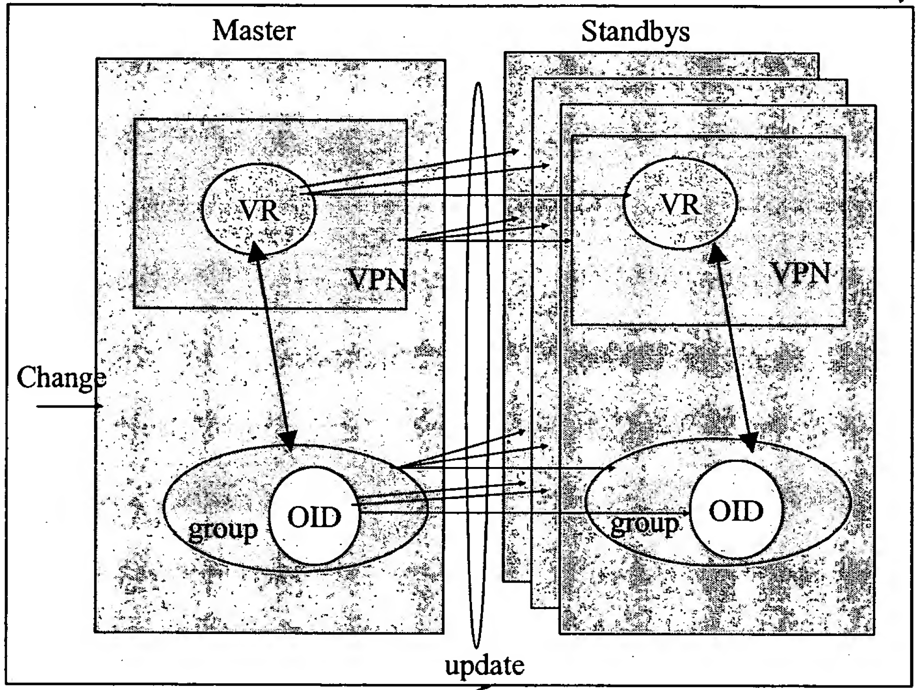


Figure ~~2~~ 25. Control Blade Redundancy

initialized to READ

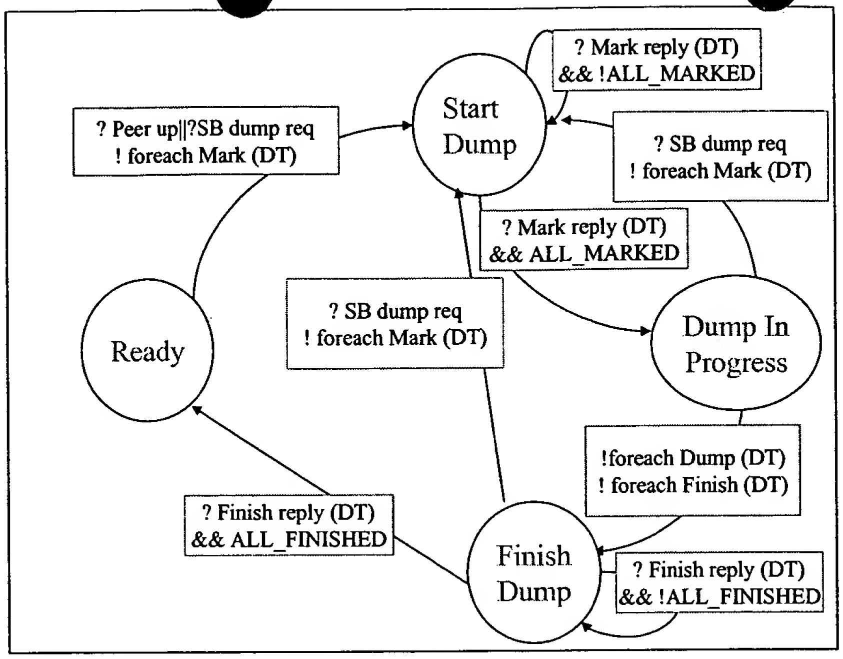


Figure ~~3~~26. CBR DUMP Master State Transition Diagram

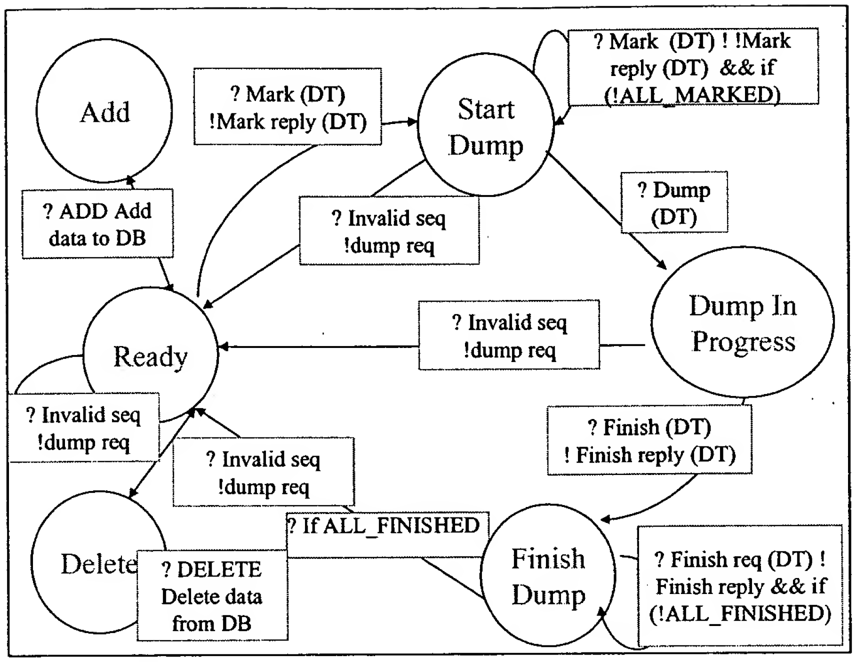

Add
? Mark (DT) !Mark reply (DT)
Start Dump
? Mark (DT) ! !Mark reply (DT) && if (!ALL_MARKED)
? ADD Add data to DB
? Invalid seq !dump req
? Dump (DT)
Ready
? Invalid seq !dump req
Dump In Progress
? Invalid seq !dump req
? Finish (DT) ! Finish reply (DT)
? Invalid seq !dump req
? If ALL_FINISHED
Delete
? DELETE Delete data from DB
Finish Dump
? Finish req (DT) ! Finish reply && if (!ALL_FINISHED)

Figure ~~4~~ 27. CBR DUMP Standby State Transition Diagram